27｜zipfile压缩库：如何给数据压缩&加密备份?

2021-04-13 尹会生

Python自动化办公实战课 进入课程 >

讲述：尹会生
时长 18:31 大小 16.97M

你好，我是尹会生。

你在日常工作中，肯定和压缩文件打过交道，它们能把文件夹制作成一个体积更小的压缩文件，不仅方便数据备份，还方便作为邮件附件来传输，或者与他人共享。

但是如果你需要每天都进行数据备份，或者把压缩包作为每天工作的日报发送给领导，你肯定希望它能自动化的压缩。面对这个需求，我们同样可以通过 python 来解决。我们可以用 Python 来自动压缩文件夹，并为压缩包设置密码，保证备份数据的安全。

在 Python 中，要想实现数据的压缩，一般可以采用基于标准库 zipfile 的方式来实现，也可以采用命令行方式来实现。

当我们希望能够用 Python 自动压缩一个无需密码保护的文件夹时，可以通过 zipfile 来实现，它的好处是使用简单，而且不用安装任何的软件包，就能制作出“zip”格式的压缩包。不过 zipfile 没法对压缩文件进行加密，因此当你需要对压缩文件加密时，还需要调用可执行命令。

这两种实现方式就是我们今天要学习的重点了，接下来我们分别看一下这两种方式的具体操作方法。

使用 zipfile 实现无密码压缩

如果我想要把“C:\data\”文件夹压缩为“当前日期.zip”文件，就可以使用**目录遍历、按日期自动生成压缩包的文件名、把文件夹写入压缩文件这三个步骤来实现。**

目录遍历

我们先来学习怎么实现目录遍历功能。我在第 16 讲已经为你讲解过它的技术细节了，这里我就继续使用 os 库来实现目录的遍历。

由于目录遍历的功能与其他功能之间的调用关系耦合得比较宽松，所以我就把目录遍历功能单独定义成一个 getAllFiles() 函数，并把要遍历的目录作为函数的参数，把该目录下的所有文件以及所在路径作为函数的返回值。

我把 getAllFiles() 函数的代码放在下方，供你参考。

复制代码

```
import os

# 遍历目录，得到该目录下所有的子目录和文件
def getAllFiles(dir):
    for root,dirs,files in os.walk(dir):
            for file in files:
                yield os.path.join(root, file)
```

细心的你一定发现了，在函数 getAllFiles() 的返回语句中，我使用 yield 语句代替了之前学习过的 return 语句返回文件路径和名称。为什么我要使用 yield 语句呢?

原因就在于，**一个函数如果使用 yield 语句来返回的话，这个函数则被称作生成器**。yield 的返回数据类型以及对类型的访问方式，都和 return 不同。我来为你解释一下 yield 和 return 的具体区别，以及使用 yield 的好处。

首先从返回类型来看，yield 返回的数据类型叫做生成器类型，这一类型的好处是调用 getAllFiles() 一次，函数就会返回一个文件路径和文件名。而 return 返回的是一个列表类型，需要一次性把要备份目录下的所有文件都访问一次，一旦要备份的文件数量非常多，就会导致计算机出现程序不响应的问题。

除了返回类型，还有调用方式也和 return 不同。使用 yield 返回的对象被称作生成器对象，该对象没法像列表一样，一次性获得对象中的所有数据，你必须使用 for 循环迭代访问，才能依次获取数据。

此外，当所有的数据访问完成，还会引发一个“StopIteration”异常，告知当前程序，这个生成器对象的内容已经全部被取出来，那么这个生成器将会在最后一次访问完成被计算机回收，这样 yield 就能够知道对象是否已经全部被读取完。

从 yield 和 return 的行为对比，可以说，yield 返回对象最大的好处是可以逐个处理，而不是一次性处理大量的磁盘读写操作，这样就有效减少了程序因等待磁盘 IO 而出现不响应的情况。这就意味着你不必在调用 getAllFiles() 函数时，因为需要备份的文件过多，而花费较长的时间等待它执行完成。

按日期自动生成压缩包的文件名

接下来我们来学习一下按日期自动生成压缩包的函数genZipfilename()。按日期生成文件名，在定时备份的场景中经常被用到，我们希望每天产生一个新的备份文件，及时保存计算机每天文件的变化。

这就要求今天的备份的文件名称不能和昨天的同名，避免覆盖上次备份的文件。

所以 genZipfilename() 函数就把程序执行的日期作为文件名来进行备份，例如当前的日期是 2021 年 4 月 12 日，那么备份文件会自动以“20210412.zip”作为文件名称。我把代码贴在下方，供你参考。

复制代码

```
import datetime

# 以年月日作为zip文件名
def genZipfilename():
    today = datetime.date.today()
    basename = today.strftime('%Y%m%d')
    extname = "zip"
    return f"{basename}.{extname}"
```

在这段代码中，“datetime.date.today()”函数能够以元组格式取得今天的日期，不过它的返回格式是元组，且年、月、日默认采用了三个元素被存放在元组中，这种格式是没法直接作为文件名来使用的。因此你还需要通过 strftime() 函数把元组里的年、月、日三个元素转换为一个字符串，再把字符串作为文件的名称来使用。

把文件夹写入压缩文件

最后，准备工作都完成之后，你就可以使用 zipfile 库把要备份的目录写入到 zip 文件了。zipfile 库是 Python 的标准库，所以不需要安装软件包，为了让这个完整脚本都不需要安装第三方软件包，我在实现文件遍历的时候同样采用 os 库代替 pathlib 库。

除了不需要安装之外，zipfile 库在使用上也比较友好，它创建和写入 zip 文件的方式就是模仿普通文件的操作流程，使用 with 关键字打开 zip 文件，并使用 write() 函数把要备份的文件写入 zip 文件。

所以通过学习一般文件的操作，你会发现 Python 在对其他格式的文件操作上，都遵循着相同的操作逻辑，这也体现出 Python 语言相比其他语言更加优雅和简单。

那么我把使用 zipfile 库实现创建 zip 文件的功能写入 zipWithoutPassword() 函数中，你可以对照一般文件的写入逻辑来学习和理解这段代码，代码如下：

复制代码

```
from zipfile import ZipFile

def zipWithoutPassword(files,backupFilename):
    with ZipFile(backupFilename, 'w') as zf:
        for f in files:
            zf.write(f)
```

对比一般的文件写入操作，zip 文件的打开使用了“ZipFile() 函数”，而一般文件的打开使用了 open 函数。写入方法与一般文件相同，都是调用“write()”函数实现写入。

这三个函数，也就是函数 getAllFiles()、genZipfilename() 和 zipWithoutPassword()，就是把备份目录到 zip 文件的核心函数了。我们以备份“C:\data”文件夹为“20210412.zip”压缩文件为例，依次调用三个函数就能实现自动备份目录了，我把调用的代码也写在下方供你参考。

复制代码

```
if __name__ == '__main__':
    # 要备份的目录
    backupDir = r"C:\data"
    # 要备份的文件
    backupFiles = getAllFiles(backupDir)
    # zip文件的名字“年月日.zip”
    zipFilename = genZipfilename()
    # 自动将要备份的目录制作成zip文件
    zipWithoutPassword(backupFiles, zipFilename)
```

在执行这段代码后，就会在代码运行的目录下产生“20210412.zip”文件，你通过计算机上的 winrar 等压缩软件查看，就会发现其中会有“C:\data”文件夹下的所有文件。由于文件名称是以当前日期自动产生的，所以每天执行一次备份脚本，就能实现按天备份指定的文件夹为压缩包了。

不过在备份时，除了要保证数据的可用性，你还有考虑数据的安全性，最好的办法就是在备份时为压缩包指定密码。接下来我就带你使用命令行调用实现有密码的文件压缩。

使用可执行命令实现有密码压缩

在制作有密码的压缩包时，我们必须使用命令代替 zipfile 来压缩文件，因为 zipfile 默认是不支持密码压缩功能的。当你需要对压缩数据有保密性的要求时，可以使用 7zip、winrar 这些知名压缩软件的命令行进加密压缩。

我在本讲中就以 7zip 压缩工具为例，带你学习一下怎么使用 Python 通过命令行方式调用 7zip 实现文件的加密压缩。

执行方式和执行参数

要想使用 7zip 实现压缩并被 Python 直接调用，你除了需要在 Windows 上安装 7zip 外，还需要知道它的**执行方式和执行的参数。**

我先来带你学习一下执行方式。7zip 软件 Windows 安装成功后，它的命令行可执行程序叫做“7z.exe”。但是它想要在命令行运行的话，需要指定程序的完整路径。例如：“c:\path\to\installed\7zip\7z.exe”。如果你希望在命令行直接输入“7z.exe”运行，需要你把可执行程序放在命令搜索路径中。我在这里有必要为你解释一下命令搜索路径的概念，有助于你以后在各种操作系统上执行命令行工具。

一条命令要想运行，必须要使用**路径 + 可执行文件的名称**才可以。例如我 Windows 中，需要把 Python 的可执行命令“python.exe”安装到“C:\python3.8\scripts\python.exe”这一位置。

那么，一般情况下当你需要运行 Python 解释器时，必须输入很长的路径。这种做法在经常使用命令行参数时没法接受的，一个是你需要记住大量命令的所在路径，另一个是较长的路径也会降低你的执行效率。

因此在各种操作系统上，都有“命令搜索路径”的概念。在 Windows 中，命令搜索路径被保存在 Path 环境变量中，Path 变量的参数是由分号分隔开的文件夹，即：当你在命令行输入“python.exe”并回车运行它时，操作系统会遍历 Path 变量参数中的每个文件夹。如果找到了“python.exe”文件，就可以直接运行它，如果没有找到，则会提示用户该命令不存在。这就避免你每次执行一条命令时都需要输入较长的路径。

再回到 7zip 的命令行执行文件“7z.exe”上，我把它安装在“C:\7zip\”文件夹下，如果你希望执行运行 7z.exe，且不输入路径，那么根据上面的分析，现在有两种解决办法。

1. 把 7z.exe 放到现有的命令搜索路径中，例如“C:\python3.8\scripts\”文件夹。
2. 把 7z.exe 所在的文件夹“C:\7zip\”加入到命令搜索路径 Path 变量的参数中。加入的方法是在 Windows 的搜索栏搜索关键字“环境变量，然后在弹出的环境变量菜单，把路径加入到 Path 变量参数即可。

设置完成环境变量后，7z.exe 就不必在命令行中输入路径，直接运行即可。

在你掌握了执行方式后，我再来带你学习一下它的参数，要想使用支持密码加密方式的 zip 压缩包，你需要使用四个参数，它们分别是：

1. a 参数：7z.exe 能够把文件夹压缩为压缩包，也能解压一个压缩包。a 参数用来指定 7z 将要对一个目录进行的压缩操作。
2. -t 参数：用来指定 7z.exe 制作压缩包的类型和名称。为了制作一个 zip 压缩包，我将把该参数指定为 -tzip，并在该参数后指定 zip 压缩包的名称。
3. -p 参数：用来指定制作的压缩包的密码。
4. “目录”参数：用来指定要把哪个目录制作为压缩包。

如果我希望把压缩包“20210412.zip”的密码制作为“password123”，可以把这四个压缩包的参数组合在一起，使用如下命令行：

复制代码

```
7z.exe a -tzip 20210412.zip -ppassword123 C:\data

```

扩展 zipfile

由于命令的参数较多，且记住它的顺序也比较复杂，所以我们可以利用 Python 的 popen() 函数，把“7z.exe”封装在 Python 代码中，会更容易使用。

因此我在无密码压缩的代码中，就可以再增加一个函数 zipWithPassword()，用来处理要压缩的目录、压缩文件名和密码参数，并通过这个函数，再去调用 popen() 函数，封装命令行调用 7z.exe 的代码，从而实现有密码的压缩功能。代码如下：

复制代码

```
import os
def zipWithPassword(dir, backupFilename, password=None):
    cmd = f"7z.exe a -tzip {backupFilename} -p{password} {dir}"
    status = os.popen(cmd)
    return status

```

我来解释一下这段代码。在实现有密码压缩的函数中，为了调用函数更加方便，我把“压缩的文件夹、zip 文件名称、密码”作为该函数的参数，这样当在你调用 zipWithPassword() 函数时，就能指定所有需要加密的文件和目录了。此外，在执行命令时，我还通过 os.popen() 函数产生了一个新的子进程（如果你不记得这个概念，可以参考第五讲）用来执行 7z.exe，这样 7z.exe 会按照函数的参数，把文件夹压缩成 zip 文件并增加密码。

通过 zipWithPassword() 函数，你就能够实现 zipfile 的扩展，实现有密码文件压缩功能了。

小结

最后，我来为你总结一下今天这节课的主要内容。我通过 zipfile 库和 7zip 软件，分别实现了无密码压缩文件和有密码压缩文件。

无密码压缩文件更加简单方便，而有密码压缩文件更加安全，配合自动根据当前日期改变压缩文件名称，可以作为你进行每日数据自动化备份的主要工具。

除了备份功能的学习外，我还为你讲解了新的函数返回方式 yield，和 return 不同的是，yield 返回的是生成器对象，需要使用 for 迭代方式访问它的全部数据。yield 语句除了可以和 zipfile 库一起实现数据备份外，还经常被应用于互联网上的图片批量下载压缩场景中。

以上内容就是怎么实现无密码和有密码压缩的全部内容了，我将完整代码贴在下方中，一起提供给你，你可以直接修改需要备份的目录，完成你自己文件夹的一键备份脚本。

复制代码

```
from zipfile import ZipFile
import os
import datetime

# 以年月日作为zip文件名
def genZipfilename():
    today = datetime.date.today()
    basename = today.strftime('%Y%m%d')
    extname = "zip"
    return f"{basename}.{extname}"

# 遍历目录，得到该目录下所有的子目录和文件
```

```
def getAllFiles(dir):
    for root,dirs,files in os.walk(dir):
            for file in files:
                yield os.path.join(root, file)

# 无密码生成压缩文件
def zipWithoutPassword(files,backupFilename):
    with ZipFile(backupFilename, 'w') as zf:
        for f in files:
            zf.write(f)

def zipWithPassword(dir, backupFilename, password=None):
    cmd = f"7z.exe a -tzip {backupFilename} -p{password} {dir}"
    status = os.popen(cmd)
    return status

if __name__ == '__main__':
    # 要备份的目录
    backupDir = "/data"
    # 要备份的文件
    backupFiles = getAllFiles(backupDir)
    # zip文件的名字“年月日.zip”
    zipFilename = genZipfilename()
    # 自动将要备份的目录制作成zip文件
    zipWithoutPassword(backupFiles, zipFilename)
    # 使用密码进行备份
    zipWithPassword(backupDir, zipFilename, "password123")
```

思考题

按照惯例，我来为你留一道思考题，如果需要备份的是两个甚至更多的目录，你会怎么改造脚本呢？

欢迎把你的想法和思考分享在留言区，我们一起交流讨论。也欢迎你把课程分享给你的同事、朋友，我们一起做职场中的效率人。我们下节课再见！

提建议

更多课程推荐

© 版权归极客邦科技所有，未经许可不得传播售卖。 页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 26 | 快速提取图片中的色块，模仿一张大师的照片

下一篇 28 | Celery库：让计算机定时执行任务，解放人力

精选留言 (2)

写留言

Soul of the Drago...
2021-04-15

老师，请问为什么我用zipWithPassword()函数对文件夹进行压缩后，生成的压缩包还是没有密码的状态呢？

展开

作者回复: zipWithPassword() 只能用于解压缩，它的详细描述写在官方文档中。所以我在课程中教你用命令行执行压缩就是这个目的了

LDxy

2021-04-13

使用 yield 返回的对象被称作生成器对象，该对象没法像列表一样，一次性获得对象中的所有数据，你必须使用 for 循环迭代访问，才能依次获取数据。这句话怎么理解呢？调用一次getAllFiles是返回一个数据还是返回所有数据？

展开

作者回复: 调用一次之后，会返回一个可迭代对象，可迭代对象默认只包含第一个元素，必须对它再次迭代，你才能获得下一个元素